Flexible Handling of Diverse Dependability Requirements in MARS *

H. Kantz
Institut für Technische Informatik
TU-Vienna,
Vienna, Austria

Abstract

Computers are used in a broad field of applications, where the delivery of application service depends on the proper operation of the controlling computer system. Moving control from human operators to computers has increased the need for dependable computers.

Depending on the type of application, the computer system has to deal with a wide spectrum of dependability requirements. It is therefore extremely important to develop flexible computer architectures allowing an easy adaptation to the demanded dependability properties.

This paper analyzes the MARS architecture for its suitability and flexibility to cope with reliability, safety, and availability requirements.

1 Introduction

Due to tremendous hardware technology developments and decreasing cost of hardware components, computers are now used in an ever increasing field of industrial applications. Hand in hand with this development, due to the ever increasing complexity of control tasks, a trend of moving control tasks from human operators to computers has arisen, thereby increasing computers' importance and criticality. In these applications a computer failure can cause severe consequences such as high additional costs (e.g. repair costs, loss of production) or even endangering human lives. Therefore, it is of utmost importance to guarantee a certain dependability behavior of the system.

Dependability requirements cover a wide area including reliability and safety aspects, availability of service, and so on. Unfortunately, there exists no general method for maximizing all these requirements concurrently and very specific design and configuration decisions, depending on the type of application, are demanded. These circumstances have spurred the need for computer architectures which can be easily tuned with respect to their dependability requirements.

In this paper we critically examine the MARS architecture [11] for its suitability to cope with different dependability requirements, namely reliability, safety, and availability. These three measures necessitate different strategies for tuning computer systems on the hardware level, on the operating system level, and on the application software level. In this paper we are concentrating on mechanisms on the hardware level and operating system level which are transparently to the application software and we do not discuss the impact of design faults and diverse software fault-tolerance techniques such as N-version Programming [3], N-Self-Checking Programming [15], or Recovery Blocks [21] on system dependability. In this paper, the main emphasis is put on an analysis of the flexibility and adaptability of such design and configuration decisions and on the comparison of the different measures and their trade-offs.

This paper is organized as follows: The next section discusses and compares the dependability measures reliability, safety, and availability and illustrates the inherent trade-offs between these different dependability attributes. Section 3 gives a brief presentation of the reference architecture MARS and strategies for its tuning. The following section presents the derived Markov models for analyzing the different architectural variants, while Section 5 discusses the numerical results. Finally, the main points of this paper are summarized in Section 6.

2 Coping with different Dependability Requirements

Computer systems are used in a large scale of different fields of applications ranging from telephone switching systems, process control applications, power plant control stations to space and avionic flight control systems. Depending on the type of application, computer

* Accepted for publication at 10th Symposium on Reliable Distributed Systems, Sep. 1991, Pisa, Italy. This work has been supported in part by the ESPRIT Basic Research Project 3092 "Predictably Dependable Computing Systems (PDCS)"

systems have to cope with various dependability requirements: For example, Federal Aviation Authority (FAA) requires for flight control systems that the probability of a failure should be less that 10^{-9} per hour for a 10 hour flight, during which no maintenance will be available [6]. British requirements for reactor safety systems mandate that no single fault shall cause a reactor trip and that there be a 10^{-7} average probability, over 5000 hours, of failure to meet a demand to trip [26]. The dependability requirements for the Bell Systems Electronic Switching System (ESS) [25] demand a down-time of the total system not exceeding 2 hours over a 40 year period, with not more than 0.02 % of calls being handled incorrectly.

In recent years, various papers dealing with modeling and evaluation of these dependability measures have been published, e.g. [2,5,8,16,18,23]. In this paper we are dealing with three basic types of dependability measures: Reliability, Safety, and Availability.

The *reliability* $R(t)$ of a system is defined as the probability that the system will perform its intended function for a specified period of time t under a set of specified environmental conditions. For better result interpretation, reliability is often expressed as the mean time of the continuous delivery of the intended behavior (i.e. the mean time to failure (MTTF)). The MTTF can be directly derived from the system's reliability: $MTTF = \int_0^\infty R(t)dt$.

The *safety* $S(t)$ of a system is defined as the probability that no catastrophic failure will happen for a specified period of time t under a set of specified environmental conditions. Similar to reliability, system safety is often expressed as the mean time to a catastrophic failure (MTCF): $MTCF = \int_0^\infty S(t)dt$.

These two definitions are not synonymous. Reliability is concerned with guaranteeing a failure free system behavior, whereas safety is concerned with the prevention of a catastrophe. Reliability has to deal with every possible error causing an interruption of service, while in safety measures only errors causing a catastrophe are considered. Errors causing only an interruption of service are of no concern here.

There are two classes of safety critical systems: Fail-operational systems and fail-safe systems. In a critical fail-operational system, like avionic systems, the system does not have a safe state which can be reached in case of a system failure. It has to function despite errors within the control system. In this class of applications it is the goal to maximize reliability and safety, but one can easily recognize that reliability and safety often imply conflicting requirements and systems cannot always be built to maximize both.

In so called fail-safe systems, in case of a failure, the system can switch into a safe state which generally implies an interruption or degradation of service. However, safety is often achieved at the cost of system's reliability.

Reliability and safety have only the system operation in mind, but do not consider repair activities and down-time intervals. Availability is a measure of the delivery of the proper service with respect to the alternation of delivery of proper and improper service. *Availability* $A(t)$ is defined as the probability that the system is operational at time t. Unlike reliability, availability evaluation does not require continuous service during the interval $(0, t)$. The system can go through the fail-repair cycle a number of times in this interval.

Availability is correlated with reliability but includes beside the failure measures, system restoration time as well. A correlation of availability and safety measures generally provides no significant results, the consequences of a catastrophe are in general so tremendous that system restoration is no longer of prime importance.

3 An Overview of MARS

3.1 The Architecture of MARS

The fault-tolerant distributed real-time system architecture MARS (**MA**intainable **R**eal-Time **S**ystem) [11] has been designed to serve as a system architecture where hard deadlines are dictated by the environment and have to be met by the system. Main emphasis is put on a deterministic timing behavior, even under anticipated peak load conditions.

A MARS system consists of a set of clusters which may be interconnected in an arbitrary topology depending on the application requirements. Each cluster consists of a set of autonomous self-contained computers, called components. Each component executes a predefined fixed set of tasks which communicate solely by the exchange of messages via a real-time local area network (MARS buses).

Each component contains a local real-time clock. A global time base is established through the use of a fault-tolerant clock synchronization algorithm. This global time is used to realize a TDMA-strategy (time division multiple access) controlling the access to the communication channels.

The MARS architecture is based on fail-silent components. They either operate as intended or do not produce any results at all. All internal failures of a component are mapped to one external failure event, the omission of messages. To achieve a high degree of fail-silent coverage extensive error detection mechanisms have been implemented. On the hardware level

watchdog timers, power supply monitoring, temperature monitoring, parity checks, and conservative hardware design are provided. On the operating system level, the use of robust storage structures, massive use of assertions, plausibility tests, and checks of the execution time of system routines are implemented. For an extensive discussion and an analysis of the benefits of these mechanisms the reader is referred to [13].

Fault tolerance is provided by actively redundant components. Due to the fail-silent property of the MARS components, the required service can be delivered as long as at least one of the redundant partner component operates correctly. A set of actively redundant partner components forms a fault-tolerant unit (FTU).

3.2 Tuning the MARS Architecture

We classify the considered types of FTU failures in two categories: Spare exhaustion failures and lack-of-coverage failures: The first class includes all types of failures caused by an exhaustion of available resources. This class of failures is mainly influenced by the components' failure rates, the repair strategy, and the degree of redundancy.

Lack-of-coverage failures are caused by a non-detection or an incorrect handling of an internal failure. These failures are mainly dependent upon the error detection strategies and their coverage values.

In the rest of this section we discuss mechanisms for tuning the MARS architecture with respect to these failure sources. We concentrate ourselves on mechanisms which can be realized without adversely affecting the user's application tasks, thus are fully transparent on the application level.

Degree of redundancy: In the basic MARS architecture two actively redundant partner components form an FTU. Additional redundancy can be added to an FTU in the form of shadow components. A shadow component receives and processes messages and therefore maintains its internal state consistent to the states of the redundant partner components. However, a shadow component does not broadcast any message on the MARS buses and no TDMA slot is assigned to it. In case of a failure of an active partner component (detected by a membership protocol [12] within 2 TDMA cycles), the shadow component immediately becomes active using the TDMA slot of the failed component. It should be noted that using the technique of shadow components MARS can handle various degrees of redundancy without affecting the system's timing behavior.

Maintenance Strategies: The restoration of faulty components has a great influence on spare exhaustion failures. To quantify this influence and to achieve a balance between maintenance costs and system's dependability we analyze different maintenance intervals ranging from 1 day up to 6 months.

Fail-silent coverage: Achieving a perfect fail-silent behavior of components is an awarding but unrealistic goal. The quantitative assessment of the degree to which the component's fail-silent assumption is verified can be represented by the notion of an assumption coverage [19]. It defines the probability that the assumed fail-silent behavior is reached. In the MARS architecture an assumption coverage close to 1 is of crucial importance. Besides the above discussed techniques for increasing the fail-silent property of the component, temporal redundancy [14] can be applied. For this purpose, an attempt is made to execute each task a second time to detect transient faults occurring during application software execution by comparing their produced results. Since tasks are scheduled statically in MARS, enough time for the worst case execution time of each task has to be reserved. To this purpose an upper bound of the task's execution time is calculated by a MAXT-analyzer [20]. Since this tool does not fully extract the semantic behavior of the task, in most cases the task's real execution time is significantly lower than the calculated upper bound. This implies that the reserved processor time is not fully utilized. This remaining processor time is used for a second, time redundant execution of the task, thus allowing the detection of transient failures. In fact, the length of the time slice for a task has to be set that it assures a certain probability that the second time redundant execution of the task will be completed. If the second task execution completes, the signatures (calculated from the task's output) of the results are compared and in case of a mismatch, the component is turned off. An increase of the time slot increases the coverage parameter, but also increases the response time of the task.

Error-detection: As elaborated in the paragraph above, there remains a small probability that a component failure is not covered and the receiver gets a wrong result. On the receiver's side a two level comparison can be performed. First, the CRC (cyclic redundancy check) field of each received message is checked to detect transmission errors, and corrupted messages are discarded. As a second step, the signatures of the correctly received messages from the redundant partner component can be compared depending on the operational mode. We distinguish two operational modes. In the so called *Safety Mode* the goal is to maximize the error detection coverage, even at the cost of continuation of system service. In order to achieve this design objective the receiver compares the signatures of the re-

dundant messages. In case of a mismatch, the receiver can no longer rely on these results, and we assume the system will enter a safe state. If only one message is received, it is assumed to be correct. In the second operational mode, termed *Reliability Mode*, each received message is assumed to be correct, and no comparison takes place. An interesting approach for weakening the rigid policy in system shut-down in the safety mode, and to guide the message selection process in the reliability mode is to attach a dependability tag (d-tag) to each message [17]. A d-tag expresses the trustworthiness of each message which can be determined through acceptance tests, consistency tests, or hardware tests as temperature monitoring or power supply monitoring. The receiver's decision is then performed according to the d-tag of the message.

4 Dependability Model of MARS

4.1 Model Assumptions

In a model, the relevant system properties are abstracted from the real system behavior, and unimportant ones are neglected. Modeling implies some kind of idealization. Abstracting the really important system behavior is based on the modeler's intuition. Therefore it is important for an appropriate interpretation of the modeling results to know the chosen assumptions and idealization. This subsection summarizes the chosen model assumptions and idealizations.

- Faults occurring in one component are statistically independent of faults occurring in other components.
- This model deals with transient and permanent failures. Each operational component contains two data structures which are called h-state (history state) and i-state (initialization state) [10], therefore we further distinguish between transient failures causing errors in the h-state and in the i-state. The h-state comprises all relevant information which has been accumulated in the sequence of computations since the initialization of the component. The h-state is dynamic and can be changed by a computation. The i-state specifies the contents of the read-only memory and is not changed during a computation. It consists of the program and permanent data of the component.

  This classification of the component's internal state leads to three different exponentially distributed component failure types:

  - Transient failures corrupting the h-state of a component with failure rate λ_h.
  - Transient failures corrupting the i-state of a component with failure rate λ_i.
  - Permanent failures with failure rate λ_p.

  Due to the membership protocol the transient bus failure rate is subsumed in the failure rates of the components. Using two actively redundant buses, the permanent bus failure rate is so small that it can be neglected.
- These three types of component failures require different repair activities:

  - If the h-state is corrupted, only the h-state of the component needs to be resynchronized. This is performed with rate μ_h.
  - A corrupted i-state requires a reboot of the i-state which is performed periodically via a maintenance component and a subsequent resynchronization of the h-state with its redundant partner component with rate μ_i.
  - For repairing permanent component failures periodic maintenance is investigated. We assume that the system is maintained periodically, exchanging all faulty components and carrying out extensive system checks, so that the system is in a fault-free state afterwards. Since replacement of components and system checks are done online, we consider only the interval between two consecutive repair activities and omit the repair time of a permanent component failure.
- All repair activities due to transient and permanent failures are assumed to be perfect.
- An operating maintenance component is required for restoring the i-state of a failed component. Since the maintenance component operates actively redundant with a shadow component, we assume that transient i-state failures can always be restored with rate μ_i.
- Since shadow components do not send any messages on the bus, we assume that a shadow component failure cannot lead to an FTU failure.
- Fault-tolerance in MARS is realized through fail-silent components. As soon as a component detects an internal error, it turns itself off. A "well-behaved" component produces either correct results or no results at all. In this model we express the probability that a component failure is covered (i.e. the faulty component is shut down properly) through a coverage parameter c. With probability $1-c$ the fail-silent behavior of the component is not achieved, and an erroneous message is broadcasted.

4.2 Model of the FTU

We model two types of FTUs. In the one variant an FTU consists of two actively redundant components, whereas in the second variant additional redundancy in form of shadow components is added. To keep the paper concise we explain the model only for the variant with shadow components. The model without shadow components can be easily derived.

In the rest of this subsection we present a Markov Graph for one FTU covering reliability, safety, and availability aspects as well. From this Markov Graph the specific Markov Graph for the desired measurements can be easily obtained in interpreting the different FTU states as described in the following.

Before starting with a detailed explanation of the Markov Model the meaning of the used abbreviations is explained. States are termed $ww|x$ or $w|xx$, $x \in \{w, h, i, p\}$, (w=working, h = transient h-state failure, i = transient i-state failure, p = permanent failure). The number of ws to the left of the bar indicates the number of working components in the FTU. The letters to the right of the bar represent the state of the inactive components.

Components whose results are used for further computation are termed *Primary Components*. States representing a Primary component's value failure are marked with a suffix $'$. The messages of the *Secondary Components* serve as the redundant ones, and the states representing a Secondary component's value failure are marked with the suffix $''$. This assignment is not statically related to a physical component. It describes only the current actual situation, and the primaries and secondaries can be changed with the progress of time, due to the operating system strategy.

Since the whole Markov Graph is complex and difficult to survey in one picture, we are splitting it in different modules and explain them separately.

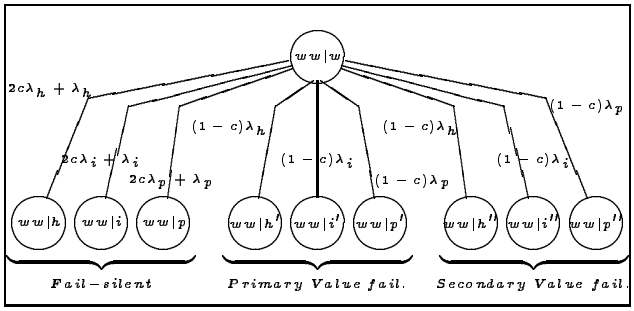

Figure 1: Markov Graph: Part 1

Figure 1 shows the state transitions from state $ww|w$ (i.e. all components are fault-free) to states with one component failure. For clarity of exposition the repair transitions with rates μ_h and μ_i are not listed here.

The nine states with one component failure within the FTU are combined into three groups: *Fail-silent*, *Primary Value Failure*, and *Secondary Value Failure*. The states of the *Fail-silent* group ($ww|h$, $ww|i$, $ww|p$) contain the states reached after a covered component failure (i.e. a working component or a shadow component fails silently). We derive a transition rate $2c\lambda_y + \lambda_y$ from state $ww|w$ to state $ww|y$, $y \in \{h, i, p\}$. The first term describes the covered failure rate of the two working components, the second term describes the failure rate of the shadow component.

With rate $(1-c)\lambda_y, y \in \{h, i, p\}$ the *Primary Value Failure* group is reached if the primary component fails with a value failure. This failure can be detected by comparison of the messages of the redundant partner components. Depending on the operational mode different actions take place: In reliability mode, in the case of a mismatch, the value of the primary is assumed to be the correct one. In this particular case the failure propagates and we pessimistically assume that the system fails. In reliability mode, these states are referred to failure states. A primary value failure in safety mode leads to the detection of the mismatch and an emergency action (i.e. safe exit). In availability mode the failure propagates. But in this operational mode we assume that this wrong result does not cause severe consequences and after the detection and shut down of the faulty component the operation continues.

The states in the third group *Secondary Value Failures* are reached with rate $(1-c)\lambda_y, y \in \{h, i, p\}$ and describe the not covered secondary value failure. In reliability and availability mode the wrong message is discarded and the secondary component is shut down. In safety mode, analogously to the primary value failure case, an emergency action is initiated. The states of this group and the primary value failure states are benign failure states in safety mode, expressing a state with a safe shut down.

In all operating states with one failed component, transient h-state (i-state) failures are recovered with rate μ_h (μ_i).

The next level of the Markov model describes the transitions from an FTU state with one failed component $(ww|y), y \in \{h, i, p\}$ to states with two failed components and vice versa.

In Figure 2 we group the FTU states with two failed component in three classes: *Fail-silent*, *Primary Value Failure*, and *Secondary Value Failure* with the same interpretation of the system states as in Figure 1. Transient failures are recovered in parallel.

Figure 3 shows the transitions from an FTU with

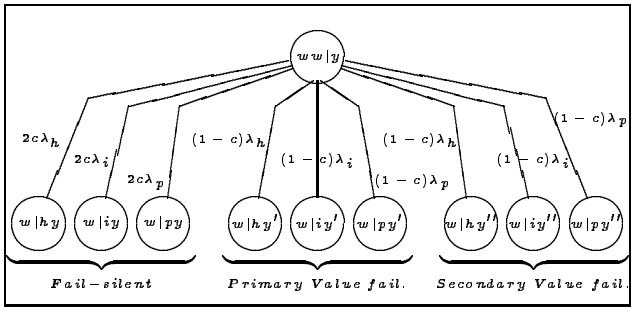

Figure 2: Markov Graph: Part 2

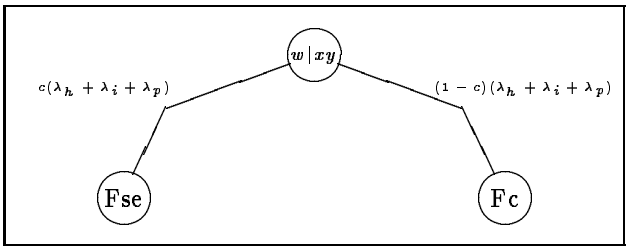

Figure 3: Markov Graph: Part 3

two failed components to an FTU where all components have failed. We distinguish between two different types of failures: Spare exhaustion and lack-of-coverage. The spare exhaustion failure state *Fse* is reached if the last operating component fails silently. If the component fails with a value failure, the lack-of-coverage state *Fc* is reached. In reliability and availability mode no distinction between the two failure states is made. In safety mode, a fail-silent failure of the component is detected in missing the component's broadcast message. So an emergency action can be initiated resulting in a benign failure. If the component fails with a value failure, we assume that it results in a catastrophic failure.

4.3 Model Parameter Values

For the model calculations we choose the following values for the failure rates, repair rates, and the coverage parameter:

- Based on our experiences with the existing MARS system we assume a permanent component failure rate $\lambda_p = 1/50000\ h^{-1}$.

  Transient failures are a dominant failure source of computer systems. Because of their sensitivity we will vary λ_t in the range from $1/3000\ h^{-1}$ to $1/5000\ h^{-1}$.

  Because of a typical size of the h-state and the i-state we choose $\lambda_h = (4/5) * \lambda_t$ and $\lambda_i = (1/5) * \lambda_t$.

- For repairing a corrupted i-state we assume a reboot rate $\mu_i = 12\ h^{-1}\ (1/5\ min^{-1})$.

  A corrupted h-state is resynchronized with rate $\mu_h = 120\ h^{-1}\ (1/30\ sec^{-1})$.

  For repair of permanent failures we investigate periodic maintenance intervals ranging from 1 day to 1 month in reliability and safety mode. In availability mode, the analyzed periodic maintenance intervals are assumed to range from 1 month to 6 months.

- A very crucial parameter of repairable computer systems is the coverage value c [4]. In the MARS architecture we have error detection mechanisms on the hardware level, operating system level, and application software level. We assume the hardware has a fail-silent coverage of 99%. We assume that 1/3 of the failures affect the operating system and 2/3 affect the application software. We further assume that the operating system can handle 90% of the failures affecting it which are not covered by the hardware, and two-fold execution of application tasks fixes 99% of the remaining failures affecting the application software. From this coverage model we derive a coverage value c of 0.9996 in the case of a two-fold execution of application tasks. In the case of a single execution of application tasks the coverage value c is 0.993. For a more detailed explanation of the error detection techniques and this coverage model the reader is referred to [13]. In its current state the coverage model is rather based on engineering judgment than on experimental evidence. However, it serves as a quantitative guideline for the current redesign of the MARS hardware and MARS operating system [24]. It is likely that after fault injection experiments [1,9] a calibration of the model becomes necessary.

5 Discussion of Numerical Results

In this paper we examine a typical MARS system configuration consisting of one cluster with 8 FTUs. For modeling the MARS cluster a two level hierarchical approach is used. At the lower level, each FTU is modeled as a Markov Graph. At the upper level, these submodels representing one FTU each, are combined via a series/parallel block diagram to form the MARS cluster. We present the calculated results of the developed reliability, safety, and availability models. The models have been evaluated with the evaluation tool SHARPE [22].

5.1 Reliability Mode

In reliability mode we investigate two different kinds of measurements. First, we analyze the probability that a mission of time t can be successfully completed. Such measures, with mission times in the order of 10 hours, are normally applied for ultra-high reliable applications, such as avionic systems [7,27]. For these applications it is assumed that the system is in a fault-free state at time $t = 0$, and that during the mission time t no replacement of permanently failed components takes place. Figure 4 shows the probability of a mission failure as a function of the mission time t.

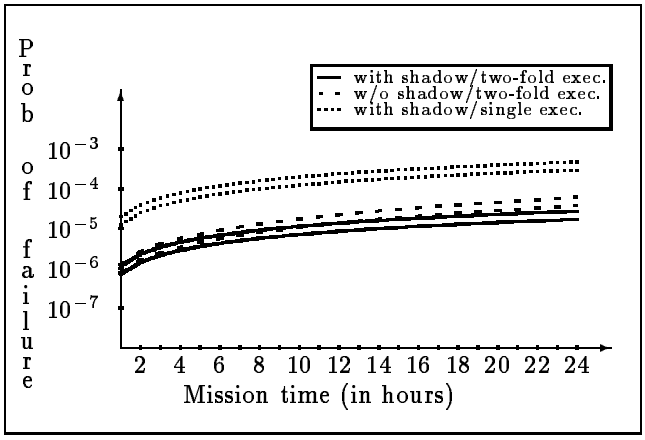

Figure 4: Probability of a mission failure

In this figure we show the numerical results for three possible MARS configurations, using the reliability operational mode for checking received messages. For each configuration a pair of curves is listed. The lower curves represent the variant with transient failure rate $\lambda_t = 1/5000\ h^{-1}$, while the upper curves represent the variant with a transient failure rate of $\lambda_t = 1/3000\ h^{-1}$.

The solid curves depict the variant with shadow components and a two-fold execution of application tasks. The architectural variant without shadow components and two-fold execution of application tasks is represented by dashed lines. These curves show that shadow components – decreasing the probability of spare exhaustion failures – have only a minor impact on the system reliability for such a short mission time. Performing only a single execution of application tasks with shadow components – increasing the probability of lack-of-coverage failures – significantly decreases system's reliability, as shown in the dotted curves.

Figure 5 shows the MTTF of the system, as a function of the periodic maintenance interval. Again, we consider the same three architectural variants, as in the previous figure, with $\lambda_t = 1/3000\ h^{-1}$ (lower curves) and $\lambda_t = 1/5000\ h^{-1}$ (upper curves).

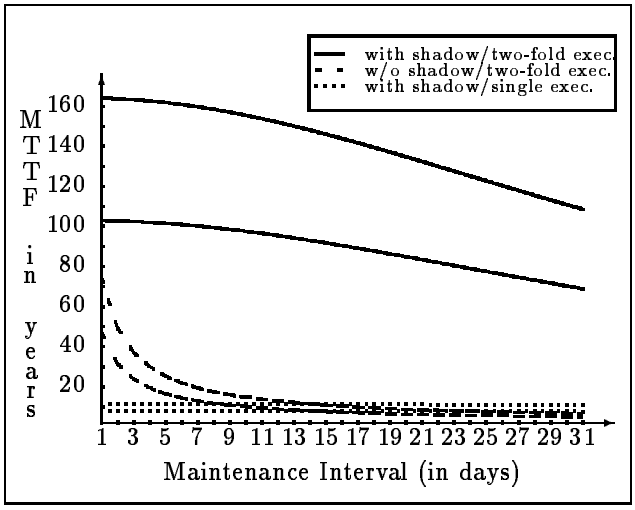

Figure 5: MTTF in reliability mode

The results reveal that for these types of measures the transient failure rate is extremely sensitive, and has significant impact on the system's reliability behavior.

The variant with shadow components and two-fold execution of application tasks allows maintenance intervals up to 1 month without significantly decreasing the MTTF. On the contrary, the architectural variant with two-fold task execution and no shadow components is only suitable for short maintenance intervals. For long maintenance intervals spare exhaustion failures become the major failure source leading to a rapid decrease of the reliability. Using shadow components and single execution of tasks leads to a short MTTF, even for short maintenance intervals. In this case lack-of-coverage failures become the dominant failure source, and even a small variation of the coverage parameter has significant impact on system's reliability. Due to space limitations a detailed parameter variation study of the coverage is only included for the MTTF in safety mode in this paper (see Section 5.2).

5.2 Safety Mode

In safety oriented applications the mean time to failure is still an important parameter, but the designer is primarily interested in the mean time to *catastrophic* failure or the probability of completing a mission without a catastrophic failure, thus the main design objective is an extremely high error detection coverage. We therefore use the safety mode for checking messages at the receiving components. In this analysis we assume that in case of the detection of an error, the system can

switch into a safe state with an interruption of service.

Again we consider three MARS architectural variants (with shadow/two-fold execution, without shadow/two-fold execution, and with shadow/single execution) with transient failure rates $\lambda_t = 1/3000\ h^{-1}$ (lower curves) and $\lambda_t = 1/5000\ h^{-1}$ (upper curves). We express the safety of a MARS application in two figures. Figure 6 shows the MTTF of the system in safety mode, and Figure 7 shows the probability that this failure has been catastrophic. With the help of the two figures one can easily derive the mean time to critical failure (MTCF).

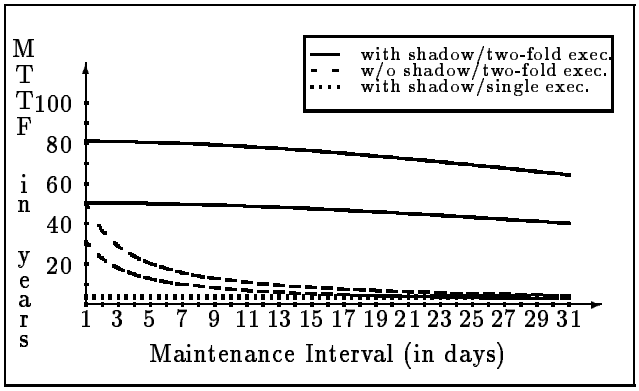

Figure 6: MTTF in safety mode

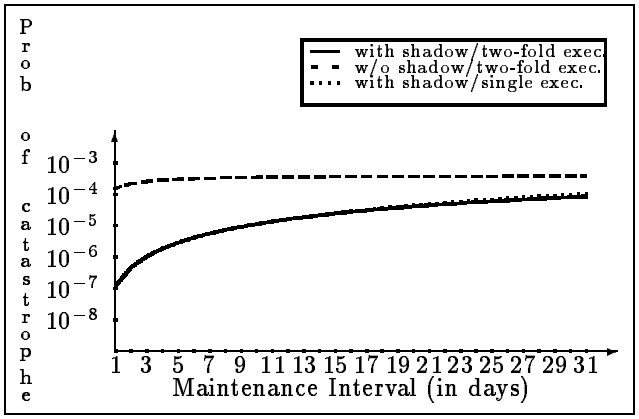

Figure 7: Probability of a catastrophic failure

The first figure allows a comparison of the MTTF between reliability and safety mode. In safety mode the MTTF of the MARS cluster decreases approximately by a factor of two. It is interesting that a variation of the transient failure rate from $1/3000\ h^{-1}$ to $1/5000\ h^{-1}$ does only minimally change the probability of an undetected, thus catastrophic failure. Therefore we show in Figure 7 only the curves for $\lambda_t = 1/5000\ h^{-1}$. The probability that the failure has been undetected is for the variant with shadow components and two-fold execution of application tasks between 10^{-7} and 10^{-5}. It results in a MTCF in the order of 10^6 to 10^8 years. Using shadow components and a single execution of application tasks results in the nearly equal probabilities. However due to the lower MTTF, the MTCF is around a factor of 10 lower than in the first variant. Using no shadow components, the probability of an undetected failure is around 10^{-4}, resulting in a MTCF in the order of 10^4 to 10^5 years.

The significant differences in the MTTF between the variant with two-fold execution and single execution in Figure 6 show the extreme sensitivity of the coverage parameter. Since the current coverage model is only of theoretical nature and not validated with experiments, we will have a closer look at the impact of the coverage parameter and perform some parameter variations for the architectural variant with shadow components and two-fold execution with failure rate $\lambda_t = 1/5000\ h^{-1}$ and $\lambda_t = 1/3000\ h^{-1}$. The coverage value $c = 0.9996$ implies that only 1 out of 2500 faults is not handled correctly, thus leading to an FTU failure. The following figure shows a variation of the coverage from $c = 0.999$ (1 out of 1000), $c = 0.999\dot{3}$ (1 out of 1500), $c = 0.9995$ (1 out of 2000) to $c = 0.9996$ (1 out of 2500). The curves are normalized with respect to the MTTF of the variant with $\lambda_t = 1/5000\ h^{-1}$ and $c = 0.9996$.

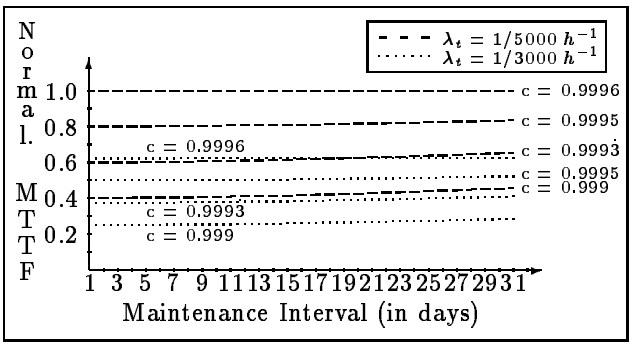

Figure 8: Normalized MTTF in safety mode

The results demonstrate the extreme importance of the coverage for the MARS architecture, and in the current redesign of the MARS hardware a high coverage is a very important design goal.

5.3 Availability Mode

In availability oriented applications the user is primarily interested in the percentage of the system's up time. Normally, in this kind of applications, the cost of the down-time is in the same order of magnitude

than the benefits of proper operation, therefore long maintenance intervals can be used. In this type of applications it is generally admissible that a small percentage of results is incorrect. We therefore assume that a single undetected failure causes only a tolerated temporarily wrong result and does not lead to a system failure, and consequently we consider only spare exhaustion failures.

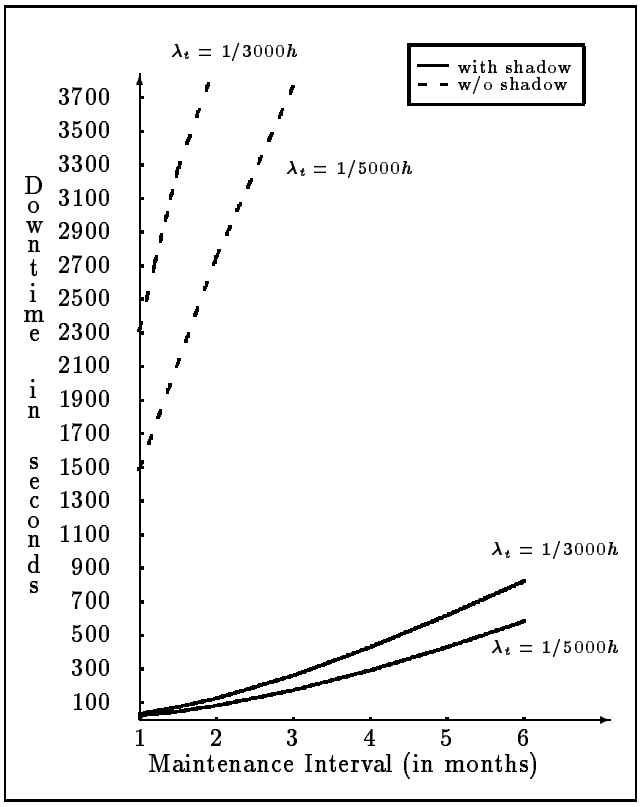

Figure 9: Downtime in seconds per year

Figure 9 shows the system's down time in seconds per year as a function of the maintenance interval, with transient failure rates $\lambda_t = 1/3000\ h^{-1}$ and $\lambda_t = 1/5000\ h^{-1}$. It is assumed that the time to reconfigure a failed MARS cluster takes 2 hours. Having no safety critical failures, we use the reliability mode for checking received messages to maximize the MTTF. The solid curves show the down-time of the variant with shadow components, and the dashed curves represent the variant without shadow components. The results show that for long maintenance intervals a MARS configuration without shadow components is unsuitable for achieving a small down-time. Adding additional redundancy drastically changes the system failure behavior allowing a significant decrease in maintenance costs by performing periodic maintenance in intervals in the order of months. Therefore, for economic considerations shadow components in conjunction with long maintenance intervals are desirable.

6 Summary and Future Work

In this paper we have analyzed variants of the MARS architecture for their handling of reliability, safety, and availability requirements. In order to provide these requirements, different strategies for tuning the MARS system have been applied: The installation of shadow components for reducing the probability of spare exhaustion failures, studies of the impact of the maintenance intervals on system's dependability, and increasing the error detection coverage by time redundant execution of application tasks and by checking signatures of received messages. It should be noted that although design and software faults play an important role in highly dependable systems, they are not included in these models. One direction of future research is extending the models in incorporating these types of faults.

The results have shown that the fail-silent coverage of components is the most sensitive parameter, and a coverage value close to 1 is a necessary prerequisite for this type of architecture. The use of general purpose hardware does not provide a sufficient fail-silent behavior, even if sophisticated error detection mechanisms at the operating system level and application software level are used. Consequently, we are currently developing a new MARS hardware where extensive error mechanisms (voltage supply monitoring, watchdog timers, memory management unit, ...) are included. Concurrently, the MARS operating system and its error detection mechanisms (robust data structures, twofold execution of application tasks, check of execution times of application tasks, ...) are redesigned to assist the hardware mechanisms to obtain the optimal benefits [24]. For the validation of these error detection mechanisms extensive fault injection experiments with a physical-fault injection tool [1] and heavy-ion radiation [9] are planned.

7 Acknowledgment

The author wishes to thank the members of the MARS research group, in particular H. Kopetz, G. Grünsteidl, P. Puschner, J. Reisinger, W. Schütz, A. Vrchoticky, and R. Zainlinger, and the anonymous referees for their constructive and valuable comments on earlier versions of this paper.

References

[1] J. Arlat, Y. Crouzet, and J. C. Laprie. Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems. In *Proc. 19th Int. Symposium on Fault-Tolerant Computing*, pages 348–355, Chicago, IL, USA, June 1989.

[2] J. Arlat and K. Kanoun. Modelling and Dependability Evaluation of Safety Systems in Control and Monitoring Applications. In *IFAC Proceedings SAFECOMP 86*, pages 157–164, Sarlat, France, Oct. 1986.

[3] A. Avizienis. The N-Version Approach to Fault-Tolerant Systems. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, Dec. 1985.

[4] W. Bouricius, W. Carter, and P. Schneider. Reliability Modeling Techniques for Self-Repairing Computer Systems. In *Proceedings of the 24th Annual ACM National Conference*, pages 295–309, 1969.

[5] C-M. Chen and J. Ortiz. Reliability Issues with Multiprocessor Distributed Database Systems: A Case Study. *IEEE Transactions on Reliability*, 38(1):153–158, Apr. 1989.

[6] Federal Aviation Administration. *System Design Analysis*. US-Department of Transportation, 1982.

[7] A. Hopkins, T. Smith, and J. Lala. FTMP - A highly reliable Fault-tolerant Multiprocessor for Aircraft. *Proceedings of the IEEE*, 66(10):1221–1239, Oct. 1978.

[8] O. Ibe, A. Sathaye, R. Howe, and K. Trivedi. Stochastic Petri Net Modeling of VAXcluster System Availability. In *Proc. 3th Int. Workshop on Petri Nets and Performance Models*, pages 112–121, Kyoto, Japan, Dec. 1989.

[9] J. Karlsson, U. Gunneflo, and J. Torin. Use of Heavy-ion Radiation from ^{252}Californium for Fault Injection Experiments. In *Int. Working Conference on Dependable Computing for Critical Applications*, Santa Barbara, CA, USA, Aug. 1989.

[10] H. Kopetz. The Failure Fault (FF) Model. In *Proc. 12th Int. Symposium on Fault-Tolerant Computing*, pages 14–17, Santa Monica, CA, USA, June 1982.

[11] H. Kopetz, A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft, and R. Zainlinger. Distributed Fault-Tolerant Real-Time Systems: The MARS Approach. *IEEE Micro*, 9(1):25–40, Feb. 1989.

[12] H. Kopetz, G. Grünsteidl, and J. Reisinger. Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System. In *Int. Working Conference on Dependable Computing for Critical Applications*, pages 167–174, Santa Barbara, CA, USA, Aug. 1989.

[13] H. Kopetz, H. Kantz, G. Grünsteidl, P. Puschner, and J. Reisinger. Tolerating Transient Faults in MARS. In *Proc. 20th Int. Symposium on Fault-Tolerant Computing*, pages 466–473, Newcastle upon Tyne, UK, June 1990.

[14] C. Krishna and A. Singh. Modelling Correlated Transient Failures in Fault-Tolerant Systems. In *Proc. 19th Int. Symposium on Fault-Tolerant Computing*, pages 374–381, Chicago, IL, USA, June 1989.

[15] J. C. Laprie, J. Arlat, C. Beounes, K. Kanoun, and C. Hourtolle. Hardware- and Software-Fault Tolerance: Definition and Analysis of Architectural Solutions. In *Proc. 17th Int. Symposium on Fault-Tolerant Computing*, pages 116–121, Pittsburgh, PA., July 1987.

[16] M. Mulazzani. Reliability versus Safety. In *IFAC Proceedings SAFECOMP 85*, pages 141–146, Como, Italy, Oct. 1985.

[17] B. Parhami. A Data-driven Dependability assurance Scheme with Applications to Data and Design Diversity. In *Int. Working Conference on Dependable Computing for Critical Applications*, pages 105–112, Santa Barbara, CA, USA, Aug. 1989.

[18] P. Pignal. An Analysis of Hardware and Software Availability exemplified on the IBM 3725 Communication Controller. *IBM Journal of Research and Development*, 32(2):268–278, Mar. 1988.

[19] D. Powell. Fault Assumptions and Assumptions Coverage. In *Second Year PDCS Report, Vol. 1*, Newcastle upon Tyne, UK, May 1991.

[20] P. Puschner and Ch. Koza. Calculating the Maximum Execution Time of Real-Time Programs. *Real-Time Systems*, 1(2):159–176, Sep. 1989.

[21] B. Randell. Design-Fault Tolerance. In J. C. Laprie A. Avizienis, H. Kopetz, editor, *The Evolution of Fault-Tolerant Computing*, pages 251–270. Springer Verlag, Vienna, Austria, 1987.

[22] R. Sahner and K. S. Trivedi. Reliability Modeling using SHARPE. *IEEE Transactions on Reliability*, 36(2):186–193, June 1987.

[23] M. Shooman and E. Cortes. Reliability and Availability Modeling of Coupled Communication Networks: A simplified Modeling Approach. In *Proc. Annual Reliability and Maintainability Symposium*, pages 129–136, Orlando, FL, USA, Jan. 1991.

[24] A. Steininger and J. Reisinger. Integral Design of Hardware and Operating System for DCCS. In *10th IFAC Workshop on Distributed Computer Control Systems*, Semmering, Austria, Sep. 1991.

[25] W. Toy. Fault-Tolerant Design of Local ESS Processors. *Proceedings of the IEEE*, 66(10):1126–1145, Oct. 1978.

[26] D. Wellbourne. Computers for reactor safety systems. *Nucl. Eng. Int.*, pages 945–950, Nov. 1974.

[27] J. Wensley, L. Lamport, J. Goldberg, M. Green, K. Levitt, P. Melliar-Smith, R. Shostak, and C. Weinstock. SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control. *Proceedings of the IEEE*, 66(10):1240–1268, Oct. 1978.